Data Compression
Lecture Notes

Dr. Faisal Aslam

Lecture 1 Introduction to Data Compression

1.1 Learning Objectives

By the end of this lecture, students will be able to:

- Understand the motivation and benefits of data compression
- Differentiate between lossless and lossy compression techniques
- Compute and interpret common compression performance metrics
- Apply the Huffman coding algorithm step by step
- Analyze real-world compression trade-offs

1.2 Introduction and Motivation: Why Compress Data?

Data compression is the process of representing information using fewer bits than its original form. It is a fundamental component of modern computing systems, enabling efficient storage, faster communication, and reduced operational costs.

Everyday applications of compression include:

- Streaming audio and video
- Image storage and sharing
- File archiving and backups
- Network communication and cloud services

Definition

Data Compression is the process of reducing the number of bits required to represent information, either:

- **Losslessly**: allowing exact reconstruction of the original data
- **Lossily**: allowing controlled loss of information to achieve higher compression

1.2.1 Benefits of Data Compression

Data compression provides three key benefits that are critical in modern computing:

1. **Reduce Storage Space**:
    - Allows more data to be stored in the same physical space
    - Enables archival of historical data that would otherwise be discarded
    - Reduces hardware requirements for storage systems
2. **Reduce Communication Time and Bandwidth**:
    - Enables faster file transfers and downloads
    - Makes high-quality streaming (4K/8K video) practical over limited bandwidth
    - Reduces latency in real-time applications like video conferencing and online gaming
    - Allows IoT devices to transmit data efficiently over wireless networks
3. **Save Money**:
    - Reduces cloud hosting costs (storage and egress fees)
    - Lowers communication costs for data transmission
    - Decreases capital expenditure on storage hardware
    - Reduces energy consumption for data centers and network infrastructure

1.3 Lossless vs. Lossy Compression

1.3.1 Lossless Compression

Lossless compression guarantees perfect reconstruction of the original data. It is essential when accuracy and data integrity are critical.

Typical applications:

- Text files and source code
- Executables and databases
- Medical, scientific, and legal data

1.3.2 Lossy Compression

Lossy compression achieves higher compression ratios by discarding information that is less perceptible or less important.

Typical applications:

- Audio (MP3, AAC)
- Images (JPEG)
- Video (H.264, H.265)

1.3.3 Choosing Between Lossless and Lossy

Factor	**Lossless Compression**	**Lossy Compression**
Reconstruction	Exact	Approximate
Data sensitivity	High	Moderate to low
Typical ratios	Low to moderate	High
Quality impact	None	Controlled degradation

Table 1: Lossless vs. Lossy Compression

1.4 Compression Performance Metrics

1.4.1 Size-Based Metrics

$$\text{Compression Ratio (CR)} = \frac{\text{Original Size}}{\text{Compressed Size}}$$

$$\text{Compression Factor} = \frac{\text{Compressed Size}}{\text{Original Size}}$$

$$\text{Space Savings (\%)} = \left(1 - \frac{\text{Compressed Size}}{\text{Original Size}}\right) \times 100\%$$

Interpretation:

- Larger compression ratios indicate better compression
- Smaller compression factors indicate better compression

1.4.2 Rate-Based Metrics

$$\text{Bits per Sample (bps)} = \frac{\text{Compressed Size (bits)}}{\text{Number of samples}}$$

$$\text{Bit-rate (bps)} = \frac{\text{Compressed Size (bits)}}{\text{Time (seconds)}}$$

These metrics are particularly important in audio and video compression systems.

1.5 Worked Example: Audio Compression Metrics

Example

Uncompressed Audio Properties

- Duration: 180 seconds
- Sampling rate: 44.1 kHz
- Bit depth: 16 bits
- Channels: 2 (stereo)

Original Size Calculation

$$\begin{aligned}\text{Total samples} &= 180 \times 44{,}100 \times 2 = 15{,}876{,}000\\ \text{Size (bits)} &= 15{,}876{,}000 \times 16 = 254{,}016{,}000\\ \text{Size (MB)} &= \frac{254{,}016{,}000}{8 \times 1{,}048{,}576} \approx 30.27\end{aligned}$$

Compression Results

Method	Size (MB)	CR	Savings	Bit-rate
FLAC (lossless)	18.16	1.67:1	40%	807 kbps
MP3 @ 320 kbps	6.75	4.49:1	77.7%	320 kbps
AAC @ 256 kbps	5.40	5.61:1	82.2%	256 kbps

1.6 Huffman Coding

Huffman coding is a widely used **lossless compression algorithm** that assigns variable-length binary codes to symbols based on their frequencies. More frequent symbols receive shorter codes.

1.6.1 Step-by-Step Huffman Coding Example

Example

Message: `MISSISSIPPI RIVER` (17 characters including space)
Symbol Frequencies

Symbol	Frequency
I	5
S	4
P	2
R	2
M	1
V	1
E	1
(space)	1

Tree Construction

1. Combine M(1) + V(1) $\rightarrow$ 2
2. Combine E(1) + (space)(1) $\rightarrow$ 2
3. Combine P(2) + R(2) $\rightarrow$ 4
4. Combine 2 + 2 $\rightarrow$ 4
5. Combine 4 + 4 $\rightarrow$ 8
6. Combine I(5) + S(4) $\rightarrow$ 9
7. Combine 8 + 9 $\rightarrow$ 17

One Possible Code Assignment

Symbol	Code	Length
I	00	2
S	01	2
P	100	3
R	101	3
M	1100	4
V	1101	4
E	1110	4
(space)	1111	4

Compressed Size

$$5(2) + 4(2) + 2(3) + 2(3) + 4(1) = 52 \text{ bits}$$

Original Size (ASCII) $= 17 \times 8 = 136$ bits
Compression Ratio $= 136/52 \approx 2.62 : 1$

1.6.2 Key Properties of Huffman Coding

- Produces prefix-free codes
- Enables instantaneous decoding
- Guarantees minimum average code length among prefix codes
- Widely used in practical compression systems

1.7 End of Chapter Questions

Exercise Lecture 1.0

Problem 1: Basic Compression Metrics

An uncompressed grayscale image has the following properties:

- Resolution: 1024×1024 pixels
- Bit depth: 8 bits per pixel

After compression, the image size is 320 KB.
Calculate:

(a) Original image size in KB

(b) Compression ratio

(c) Compression factor

(d) Space savings percentage

Exercise Lecture 1.1

Problem 2: Audio Bit-rate and Storage

A mono audio recording has the following parameters:

- Duration: 5 minutes
- Sampling rate: 48 kHz
- Bit depth: 16 bits

The file is compressed using a lossy codec to a constant bit-rate of 192 kbps. Calculate:

(a) Size of the uncompressed audio file in MB

(b) Size of the compressed file in MB

(c) Compression ratio

(d) Bits per sample after compression

Exercise Lecture 1.2

Problem 3: Comparing Compression Options

A video clip has an uncompressed data rate of 120 Mbps. Three compression options are available:

Option	Compressed Bit-rate
A	6 Mbps
B	3 Mbps
C	1.5 Mbps

For each option, calculate:

(a) Compression ratio

(b) Data consumed for a 10-minute video (in MB)

Which option would you choose for:

(i) Live video streaming?

(ii) Archival storage?

Briefly justify your answers.

Exercise Lecture 1.3

Problem 4: Huffman Coding Construction

Given the following symbol frequencies:

Symbol	Frequency
A	10
B	8
C	6
D	5
E	4
F	3
G	2
H	2

(a) Construct the Huffman tree step by step

(b) Assign a binary code to each symbol

(c) Compute the total number of bits required to encode the message

(d) Calculate the average number of bits per symbol

Exercise Lecture 1.4

Problem 5: Fixed-Length vs. Huffman Coding

Using the symbol set from Problem 4:

(a) Determine the minimum fixed-length code required

(b) Compute the total number of bits using fixed-length coding

(c) Compare the result with Huffman coding

(d) Calculate the percentage reduction in total bits achieved by Huffman coding

Exercise Lecture 1.5

Problem 6: Text Compression Scenario

A text file contains 50,000 characters and is stored using 8-bit ASCII encoding. After compression using a lossless algorithm, the file size becomes 18 KB. Calculate:

(a) Original file size in KB

(b) Compression ratio

(c) Compression factor

(d) Space savings percentage

Explain why compression ratios for text files vary significantly depending on content.

Exercise Lecture 1.6

Problem 7: Practical Design Question

You are designing a compression system for a wearable health-monitoring device that:

- Records sensor data continuously
- Has limited storage capacity
- Requires exact data reconstruction
- Operates on a low-power processor

(a) Should the system use lossless or lossy compression? Explain.

(b) Which performance metrics are most important in this scenario?

(c) Would a variable-length coding scheme be appropriate? Why or why not?

Lecture 2 Shannon's Source Coding Theorem and Huffman Coding

2.0.1 Basic Terminology and Notation

To avoid confusion, we clarify the fundamental terms used throughout information theory and data compression.

Definition

Alphabet

An *alphabet* $\mathcal{X}$ is a finite set of possible symbols. Examples:

- Binary alphabet: $\mathcal{X} = \{0, 1\}$
- English letters: $\mathcal{X} = \{\texttt{A}, \ldots, \texttt{Z}\}$
- Bytes: $\mathcal{X} = \{0, 1, \ldots, 255\}$

Definition

Symbol

A *symbol* is a single element drawn from an alphabet. For example, the letter `E` is a symbol from the English alphabet.

Definition

Random Variable

A *random variable* X is a mathematical model of a source that produces symbols. It assigns probabilities to symbols in the alphabet:

$$P(X = x), \quad x \in \mathcal{X}$$

Entropy is defined on random variables, not directly on symbols themselves.

Definition

Source

A *source* is a process that generates a sequence of symbols $(X_1, X_2, X_3, \ldots)$ according to some probability law. In this lecture, we assume discrete sources unless stated otherwise.

Definition

Message (or Sequence)

A *message* is a finite sequence of symbols generated by the source:

$$x^n = (x_1, x_2, \ldots, x_n)$$

Compression algorithms operate on messages, not on individual symbols.

Definition

Code and Codewords

A *code* assigns a binary string (codeword) to each symbol or message.

- Source symbols → codewords (e.g., Huffman coding)
- Messages → bitstreams (e.g., arithmetic coding)

Definition

Block Length

The *block length* n is the number of source symbols grouped together and encoded as a unit. Larger block lengths generally allow better compression but increase delay and complexity.

Definition

Model

A *model* estimates the probabilities of symbols or sequences. Better models lead to better compression by reducing uncertainty.

2.1 Information and Redundancy: The Core Concepts

2.1.1 Information: A Formal Measure of Uncertainty Reduction

In common usage, "information" is often conflated with data, messages, or symbols. In **information theory**, information is defined rigorously as a **quantitative measure of the reduction in uncertainty** that results from observing the outcome of a random event.

The fundamental principle, established by Claude Shannon (1948), is:

> The information gained from an event is inversely related to its probability of occurrence. Highly probable events are unsurprising and convey little information; improbable events are surprising and convey substantial information.

Definition. Let X be a random event that occurs with probability $p = \Pr(X)$. The **information content** (or *self-information*) $I(X)$ provided by the occurrence of X is defined as:

$$I(X) = \log_b \left(\frac{1}{p}\right) = -\log_b(p)$$

where:

- The base b of the logarithm determines the unit of information.

- $b = 2$ yields **bits** (binary digits).
- $b = e$ yields **nats** (natural units).
- $b = 10$ yields **hartleys** or **dits**.

The choice of base is a convention; the underlying concept is scale-invariant.

This definition captures an important intuition: as an event becomes more predictable ($p \to 1$), its information content approaches zero.

Example

Predictability vs. Information:

- In a city where it rains every day, the statement "It rained today" conveys almost no information because it was already expected.
- A file that contains only the bit '1' provides very little information, since after seeing a few bits, the rest of the file can be predicted with certainty.
- A coin that always lands heads produces outcomes, but no information, because there is no uncertainty to resolve.

Key idea: Perfect predictability implies zero information gain.

Example

Daily Weather Forecast — Information Content:

- Sunny in Phoenix (probability 0.9): $I = -\log_2 0.9 \approx 0.15$ bits
- Snow in Phoenix (probability 0.001): $I = -\log_2 0.001 \approx 9.97$ bits
- Rain in Seattle (probability 0.3): $I = -\log_2 0.3 \approx 1.74$ bits

Interpretation: Rare events carry more information because they reduce uncertainty the most. Snow in Phoenix reveals far more about the weather system than another sunny day.

In summary, information is not about how much data is observed, but about how much uncertainty is removed. This distinction between information and predictability forms the foundation of redundancy, entropy, and data compression.

2.1.2 Redundancy: The Enemy of Information and the Friend of Compression

Redundancy refers to predictable or repeated structure in data. From the perspective of information theory, redundancy is the *enemy of information* because it does not reduce

uncertainty. However, from the perspective of data compression, redundancy is a *valuable resource*: it is precisely what allows data to be represented using fewer bits.

Compression algorithms work by identifying, modeling, and removing redundancy while preserving the underlying information (in lossless compression) or perceptually important information (in lossy compression).

Redundancy appears in several common forms:

1. **Spatial Redundancy**: Neighboring data values are highly correlated.

   Example

   In a photograph of a clear blue sky, most neighboring pixels have nearly identical color values.

   - **Naive**: Store the RGB value of each pixel independently.
   - **Smarter**: Encode repeated pixel values using run-length encoding.
   - **Even smarter**: Predict each pixel from its neighbors and encode only the small prediction error.

2. **Statistical Redundancy**: Some symbols occur far more frequently than others.

   Example

   English letter frequencies:

Letter	Frequency	Letter	Frequency
E	12.7%	Z	0.07%
T	9.1%	Q	0.10%
A	8.2%	J	0.15%

   - **Inefficient**: Fixed-length coding (5 bits per letter).
   - **Efficient**: Variable-length coding (e.g., Huffman coding), where frequent letters get shorter codes.

   This reduces the average bits per letter from 5 to approximately 4.1.

3. **Knowledge Redundancy**: Information already known to both encoder and decoder.

   Example

   Medical Imaging: Both the encoder and decoder know the image represents a chest X-ray.

- The general structure of lungs and bones does not need to be encoded explicitly.
- Anatomical models can be used to predict expected structures.
- Bits can be concentrated on unexpected or diagnostically important regions.

4. **Perceptual Redundancy**: Information that humans cannot perceive.

Example

Audio Compression (MP3):

- **Frequency masking**: Loud sounds mask nearby frequencies.
- **Temporal masking**: Loud sounds mask quieter sounds before or after them.
- **Result**: A large fraction of the audio data can be discarded without perceptible loss in quality.

2.1.3 What is Entropy? Different Perspectives

The term "entropy" appears in multiple fields (thermodynamics, information theory, statistics) with related but distinct meanings. In information theory, we primarily discuss **Shannon Entropy**, named after Claude Shannon who founded the field in 1948. While there are other entropy measures (like Kolmogorov-Sinai, Rényi, and Tsallis entropies in various contexts), Shannon entropy is the foundational concept for data compression.

Definition

Shannon Entropy of a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$ having probabilities $\{p_1, p_2, \ldots, p_n\}$:

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \quad \text{bits}$$

Two Complementary Interpretations:

1. **Average Information Content**: When a symbol with probability p_i occurs, it conveys $-\log_2 p_i$ bits of information (rare events tell us more). Entropy is the *expected value* or average of this information content across all symbols.

2. **Uncertainty or Surprise**: Entropy measures how uncertain we are about the next symbol before observing it. Higher entropy means more unpredictability.

These interpretations are two sides of the same coin: *The average information gained equals the uncertainty removed by observation.*

2.1.4 Calculating Entropy: Step by Step

Let's examine both interpretations through detailed calculations:

Example

Binary Source Example - Detailed Calculation:
Consider a biased coin: P(Heads) = 0.8, P(Tails) = 0.2

Step 1: **Calculate individual information content**:

$$I_H = -\log_2(0.8) \approx 0.3219 \text{ bits}$$
$$I_T = -\log_2(0.2) \approx 2.3219 \text{ bits}$$

Interpretation: Tails (rarer event) carries more information.

Step 2: **Calculate entropy as expected value**:

$$H = 0.8 \times 0.3219 + 0.2 \times 2.3219 = 0.7219 \text{ bits}$$

Step 3: **Verify using direct formula**:

$$H = -[0.8 \log_2(0.8) + 0.2 \log_2(0.2)] \approx 0.7219 \text{ bits}$$

Key Insights:

- **Average information**: Each flip gives 0.72 bits of information on average
- **Uncertainty**: We're 72% as uncertain as with a fair coin
- **Extreme cases**:
  - Fair coin (P=0.5): $H = 1.0$ bit (maximum uncertainty/information)
  - Always heads (P=1.0): $H = 0$ bits (no uncertainty, no information)
  - 90% heads: $H \approx 0.469$ bits (less uncertainty than 80% case)

Mathematical Properties of Entropy:

- **Non-negativity**: $H(X) \geq 0$, with equality only when one outcome is certain
- **Maximum value**: For n symbols, maximum entropy is $\log_2 n$, achieved when all probabilities are equal ($p_i = 1/n$)

- **Concavity**: Entropy is a concave function of probabilities

2.1.5 Entropy of English Text: A Practical Case Study

Example

Calculating English Letter Entropy:
Based on letter frequencies in typical English text:

Letter	**Probability** (p_i)	$-\log_2 \mathbf{p_i}$	**Contribution** ($p_i \times -\log_2 p_i$)
E	0.127	2.98	0.378
T	0.091	3.46	0.315
A	0.082	3.61	0.296
O	0.075	3.74	0.281
I	0.070	3.84	0.269
N	0.067	3.90	0.261
S	0.063	3.99	0.251
H	0.061	4.04	0.246
R	0.060	4.06	0.244
D	0.043	4.54	0.195
⋮	⋮	⋮	⋮
Z	0.0007	10.48	0.007
Total	1.0		**4.18 bits**

Layered Interpretation:

- **First-order entropy (letters independent)**: 4.18 bits/letter
- **Why not 5 bits?** Because letters are not equally likely
- **Actual uncertainty is lower**: Letters have dependencies (Q is usually followed by U)
- **Comparison with encoding schemes**:

Encoding Method	**Bits/Letter**	**Efficiency**
Naive (5 bits for 26 letters)	5.00	83.6%
Huffman (letter-based)	4.30	97.2%
Using digram frequencies	3.90	107.2%*
Using word frequencies	2.30	181.7%*
Optimal with full context	1.50	278.7%*

*Percentages $>$ 100% show compression better than first-order entropy by exploiting dependencies

2.1.6 Beyond First-Order Entropy: The Full Picture

Real-world data sources exhibit strong statistical dependencies between symbols. To capture these dependencies, entropy is defined not only for single symbols, but also for sequences of symbols.

Higher-Order Entropies quantify uncertainty while accounting for increasing context:

- **Zero-order entropy (H_0)**:
$$H_0 = \log_2 |\mathcal{X}|$$
Assumes all symbols in the alphabet $\mathcal{X}$ are equally likely and independent.

- **First-order entropy (H_1)**:
$$H_1 = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$$
Accounts for symbol frequencies, but ignores dependencies between symbols.

- **Second-order entropy (H_2)**:
$$H_2 = -\sum_{x,y} p(x,y) \log_2 p(x|y)$$
Accounts for dependencies between adjacent symbol pairs.

- **Nth-order entropy (H_N)**:
$$H_N = -\sum p(x_1, \ldots, x_N) \log_2 p(x_N \mid x_1, \ldots, x_{N-1})$$
Captures dependencies across blocks of length N.

As the context length increases, the entropy per symbol typically decreases, reflecting increased predictability.

2.1.7 Entropy Rate

The **entropy rate** of a source is defined as the limiting uncertainty per symbol when arbitrarily long contexts are available:

$$H_\infty = \lim_{N \to \infty} H_N$$

For stationary ergodic sources, this limit exists and characterizes the *true information content per symbol.*

Example (English Text):

- First-order entropy: ≈ 4.0 bits/letter
- Higher-order models reduce entropy significantly
- Estimated entropy rate: ≈ 1.0–1.5 bits/letter

This large gap highlights the importance of modeling long-range dependencies.

2.1.8 The Entropy Theorem: Why It Matters

Important

Shannon's Source Coding Theorem (Informal and Formal Statement)
Let a discrete memoryless source have entropy H.

1. **Converse (Impossibility)**: No lossless coding scheme can achieve an average code length $L < H$ bits per symbol.
2. **Achievability (Possibility)**: For any $\epsilon > 0$, there exists a coding scheme with
$$H \leq L < H + \epsilon$$
for sufficiently large block sizes.

Key Implications for Compression:

- **Fundamental limit**: Entropy is a lower bound on achievable lossless compression
- **Optimality criterion**: A compression algorithm is good if L is close to H
- **Redundancy**: Any excess over H represents inefficiency

2.1.9 Practical Interpretation: English Text Compression

- **Impossible**: Average rate below the entropy rate (~ 1.0–1.5 bits/letter)
- **Naïve encoding**: 8 bits/letter (ASCII)
- **Practical compression**: 2–3 bits/letter (modern context-based compressors)
- **Theoretical limit**: Entropy rate (100% efficiency)

2.1.10 Why the Entropy Limit Is Rarely Reached Exactly

Even optimal algorithms cannot generally reach H_∞ exactly due to:

- Finite block lengths in practical implementations

- Computational and memory constraints
- Integer-length codeword restrictions (e.g., Huffman coding)
- Imperfect probabilistic models of the source

2.1.11 Key Takeaways

- Entropy measures both **average information** and **uncertainty**
- Higher-order models reduce entropy by exploiting dependencies
- The entropy rate represents the ultimate compression limit
- Shannon's theorem precisely separates the *possible* from the *impossible*
- Good compression algorithms approach the entropy rate from above

2.2 The Compression Pipeline: How Compressors Actually Work

Most compressors follow this two-stage process:

Compression Pipeline:

Input Data $\longrightarrow$ **Modeling Stage** $\longrightarrow$

Analyzes patterns — Builds probability model

$\longrightarrow$ **Coding Stage** $\longrightarrow$

Converts to bits — Using entropy coding

$\longrightarrow$ **Compressed Data**

Two-Stage Compression Pipeline:

- **Modeling Stage**: Analyzes data patterns and builds probability model
- **Coding Stage**: Converts symbols to bits using entropy coding (Huffman, Arithmetic, ANS)

2.3 Important Terminology and Concepts

2.3.1 Key Definitions with Examples

- **Symbol**: The basic unit being compressed

Example

Different domains use different symbols:

  - Text: Characters (bytes)

- Images: Pixels (RGB triples)
- Audio: Samples (16-bit integers)
- Video: Macroblocks (16×16 pixel regions)

- **Alphabet**: Set of all possible symbols

Example

- English text: 256 possible bytes (ASCII/UTF-8)
- Binary data: 256 possible byte values
- DNA sequences: 4 symbols {A, C, G, T}
- Black-white image: 2 symbols {0=black, 1=white}

- **Prefix Code**: Crucial for instant decoding

Example

Why prefix codes matter:

- Good: A=0, B=10, C=110, D=111
- "010110" decodes unambiguously: A(0) B(10) C(110)
- Bad: A=0, B=1, C=01 (not prefix-free)
- "01" could be AB or C - ambiguous!

2.3.2 The Fundamental Insight

Important

The Core Principle of Compression:

- **Random data cannot be compressed**: Maximum entropy = no redundancy
- **Real-world data is not random**: Contains patterns, structure, predictability
- **Compression finds and exploits these patterns**

  Example - Encryption vs Compression:

  - Encrypted data looks random (high entropy)
  - Compressing encrypted data gives little or no savings
  - Always compress **before** encrypting, not after!
  - Rule: Encrypt → High entropy → No compression
  - Rule: Compress → Lower entropy → Then encrypt

Important

Big Picture:

- **Entropy** = Theoretical limit of lossless compression
- **Kraft inequality** = Feasibility condition for prefix codes
- **Huffman coding** = Optimal prefix code construction
- **Block coding** = Approach entropy by coding symbols in blocks
- **Integer constraint** = Source of the "+1" overhead in Shannon's theorem
- **Arithmetic coding** = Removes integer constraint using fractional bits

Three Key Coding Methods:

Method	Purpose	Key Propert
Shannon coding	Proof of achievability	$\ell_i = \lceil -\log_2 p_i$
Huffman coding	Optimal prefix code	Minimizes exp
Arithmetic coding	Near-optimal compression	Fractional bits

2.4 Mathematical Preliminaries and Notation

Let X be a discrete random variable taking values in alphabet $\mathcal{X} = \{x_1, x_2, \ldots, x_m\}$ with probability mass function $p(x) = \Pr(X = x)$.

Definition

Source Code: A mapping $C : \mathcal{X} \to \mathcal{D}^*$ where $\mathcal{D} = \{0, 1\}$ is the code alphabet, and $\mathcal{D}^*$ is the set of all finite binary strings. The code C assigns to each symbol x_i a codeword c_i of length $\ell_i = |c_i|$.

2.4.1 Expected Code Length

For a source with probabilities $p_1, p_2, \ldots, p_m$ and corresponding codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$, the expected code length is:

$$L(C) = \mathbb{E}[\ell(X)] = \sum_{i=1}^{m} p_i \ell_i$$

2.5 Shannon's Source Coding Theorem: Formal Statement

Definition

Shannon's Source Coding Theorem (1948):

- **Converse (Lower Bound)**: For any uniquely decodable code C for a discrete memoryless source X with entropy $H(X)$, the expected length satisfies:

$$L(C) \geq H(X)$$

- **Achievability (Upper Bound)**: There exists a uniquely decodable code C such that:

$$L(C) < H(X) + 1$$

- **Block Coding**: For the nth extension of the source, there exists a uniquely decodable code C_n such that:

$$\frac{1}{n} L(C_n) \to H(X) \quad \text{as } n \to \infty$$

2.5.1 Interpretation and Significance

- **Fundamental Limit**: $H(X)$ bits/symbol is the **asymptotic lower bound** for lossless compression

- **Achievability**: We can get arbitrarily close to this limit by coding in blocks
- **Penalty Term**: The "+1" represents overhead from integer codeword lengths
- **The Integer Constraint**: Codeword lengths must be integers, but ideal lengths $-\log_2 p_i$ are typically not integers
- **Important**: The "+1" term is **not inefficiency of Huffman**, but a fundamental limitation of integer-length codes

Example

Binary Source Analysis: Consider a binary source with $P(0) = p$, $P(1) = 1 - p$:

- Entropy: $H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$
- For $p = 0.1$: $H(0.1) \approx 0.469$ bits/symbol
- Theorem guarantees: $0.469 \leq L < 1.469$ bits/symbol
- Simple code: 0→0, 1→1 gives $L = 1$ bit/symbol (efficiency 46.9%)
- Block coding can approach 0.469 bits/symbol

2.6 Code Classification and Properties

2.6.1 Hierarchical Classification of Codes

Hierarchy of Codes:

All Codes
↓
Non-singular Codes
↓
Uniquely Decodable Codes
↓
Prefix Codes

2.6.2 Formal Definitions

1. **Non-singular**: $C(x_i) \neq C(x_j)$ for $i \neq j$
2. **Uniquely Decodable**: Every finite concatenation of codewords can be decoded in exactly one way
3. **Prefix (Instantaneous)**: No codeword is a prefix of another

Important

Hierarchy Theorem: Every prefix code is uniquely decodable, but not vice versa. However, for every uniquely decodable code, there exists a prefix code with the same codeword lengths (McMillan's theorem).

Example

Code Classification Examples:

Code	Mapping	Singular?	Uniquely Decodable?	Prefix?
C_1	a→0, b→0, c→1	Yes	No	No
C_2	a→0, b→01, c→11	No	No	No
C_3	a→0, b→01, c→011	No	No	No
C_4	a→0, b→10, c→110	No	Yes	Yes

Table 2: Classification of different codes for alphabet $\{a, b, c\}$

Analysis of C_3: The string "011" is ambiguous: it could be parsed as "ab" (0 followed by 11) or as "c" (011). Since there exists a string with multiple valid parsings, C_3 is not uniquely decodable.

2.7 Kraft-McMillan Inequality: Mathematical Foundation

Theorem 1 (Kraft-McMillan Inequality)**.** *For any prefix code (or more generally, any uniquely decodable code) with codeword lengths* $\ell_1, \ell_2, \ldots, \ell_m$ *over a* D*-ary alphabet:*

$$\sum_{i=1}^{m} D^{-\ell_i} \leq 1$$

where D *is the size of the code alphabet (2 for binary).*

2.7.1 Proof Sketch for Binary Prefix Codes

1. Consider a complete binary tree of depth $L = \max_i \ell_i$
2. Each codeword of length ℓ_i occupies $2^{L-\ell_i}$ leaf positions
3. Total occupied positions: $\sum_{i=1}^{m} 2^{L-\ell_i} \leq 2^L$
4. Dividing by 2^L: $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$

Note: The tree-based proof above applies to prefix codes; the extension to uniquely decodable codes follows from McMillan's inequality.

2.7.2 Converse: Constructing Codes from Lengths

Theorem 2 (Kraft Inequality Converse)**.** *If integers $\ell_1, \ell_2, \ldots, \ell_m$ satisfy $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$, then there exists a binary prefix code with these lengths.*

Example

Verifying Kraft Inequality:

1. Consider lengths $\{1, 2, 3, 3\}$:

$$\sum 2^{-\ell_i} = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 0.5 + 0.25 + 0.125 + 0.125 = 1$$

   A prefix code exists (e.g., 0, 10, 110, 111)

2. Consider lengths $\{1, 1, 2\}$:

$$\sum 2^{-\ell_i} = 2^{-1} + 2^{-1} + 2^{-2} = 0.5 + 0.5 + 0.25 = 1.25 > 1$$

   No prefix code exists with these lengths! This violates the Kraft inequality.

2.8 Optimal Code Lengths and Shannon Coding

2.8.1 Shannon's Length Assignment

For a source with probabilities p_i, Shannon proposed the length assignment:

$$\ell_i = \lceil -\log_2 p_i \rceil$$

where $\lceil x \rceil$ is the ceiling function.

Theorem 3. *The lengths $\ell_i = \lceil -\log_2 p_i \rceil$ satisfy the Kraft inequality.*

Proof. Since $\ell_i \geq -\log_2 p_i$, we have $-\ell_i \leq \log_2 p_i$, so:

$$2^{-\ell_i} \leq p_i \quad \Rightarrow \quad \sum_{i=1}^{m} 2^{-\ell_i} \leq \sum_{i=1}^{m} p_i = 1$$

□

Example

Shannon Coding Example: Source with probabilities $\{0.4, 0.3, 0.2, 0.1\}$

1. Compute ideal lengths: $-\log_2 p_i = \{1.32, 1.74, 2.32, 3.32\}$

2. Ceiling gives: $\ell_i = \{2, 2, 3, 4\}$

3. Check Kraft: $2^{-2}+2^{-2}+2^{-3}+2^{-4} = 0.25+0.25+0.125+0.0625 = 0.6875 \leq 1$

4. Expected length: $L = 0.4 \times 2 + 0.3 \times 2 + 0.2 \times 3 + 0.1 \times 4 = 2.4$ bits/symbol

5. Entropy: $H = 1.846$ bits/symbol

6. Efficiency: $\eta = 1.846/2.4 = 76.9\%$

2.9 Detailed Proof of Shannon's Theorem

2.9.1 Lower Bound: $L \geq H(X)$

Proof. Let p_i be symbol probabilities and ℓ_i be codeword lengths of a uniquely decodable code. From Kraft-McMillan:

$$\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$$

Define $r_i = 2^{-\ell_i} / \sum_{j=1}^{m} 2^{-\ell_j}$, so $\{r_i\}$ is a probability distribution.

Using the non-negativity of KL-divergence:

$$D(p\|r) = \sum_{i=1}^{m} p_i \log_2 \frac{p_i}{r_i} \geq 0$$

Substituting r_i:

$$\sum_{i=1}^{m} p_i \log_2 p_i - \sum_{i=1}^{m} p_i \log_2 2^{-\ell_i} + \sum_{i=1}^{m} p_i \log_2 \left(\sum_{j=1}^{m} 2^{-\ell_j} \right) \geq 0$$

Since $\sum_{j=1}^{m} 2^{-\ell_j} \leq 1$, the last term is ≤ 0, giving:

$$-H(X) + \sum_{i=1}^{m} p_i \ell_i \geq 0 \quad \Rightarrow \quad L \geq H(X)$$

□

2.9.2 Upper Bound: $L < H(X) + 1$

Proof. Choose $\ell_i = \lceil -\log_2 p_i \rceil$. Then:

$$-\log_2 p_i \leq \ell_i < -\log_2 p_i + 1$$

Multiply by p_i and sum over i:

$$-\sum_{i=1}^{m} p_i \log_2 p_i \leq \sum_{i=1}^{m} p_i \ell_i < -\sum_{i=1}^{m} p_i \log_2 p_i + \sum_{i=1}^{m} p_i$$

$$H(X) \leq L < H(X) + 1$$

□

2.9.3 The Integer Length Constraint

The "+1" term in Shannon's theorem arises from the integer constraint on codeword lengths. For a **dyadic source** where all probabilities are of the form $p_i = 2^{-k_i}$ for integers k_i, we have $-\log_2 p_i = k_i$, which are integers. In this special case, we can achieve $L = H$ exactly.

2.10 Extended Source Coding and Block Codes

2.10.1 The nth Extension of a Source

For a discrete memoryless source X, the nth extension $X^n = (X_1, X_2, \ldots, X_n)$ has:

$$H(X^n) = nH(X)$$

Applying Shannon's theorem to X^n gives a code C_n with:

$$nH(X) \leq L(C_n) < nH(X) + 1$$

Thus, the average length per symbol satisfies:

$$H(X) \leq \frac{L(C_n)}{n} < H(X) + \frac{1}{n}$$

Example

Block Coding Improvement: Binary source with $p(0) = 0.9$, $p(1) = 0.1$, $H = 0.469$ bits/symbol

- Single symbol coding: Best code gives $L = 1$ bit/symbol (efficiency 46.9%)
- **Block coding with** $n = 2$: Consider coding pairs of symbols:

$$\begin{aligned} 00 &: (0.9)^2 = 0.81 \\ 01 &: 0.9 \times 0.1 = 0.09 \\ 10 &: 0.1 \times 0.9 = 0.09 \\ 11 &: (0.1)^2 = 0.01 \end{aligned}$$

- Applying Shannon coding: $\ell_i = \lceil -\log_2 p_i \rceil$ gives lengths $\{1, 4, 4, 7\}$
- Expected length: $L_2 = 0.81 \times 1 + 0.09 \times 4 + 0.09 \times 4 + 0.01 \times 7 = 1.6$ bits/block

- Per symbol: $L_2/2 = 0.8$ bits/symbol (efficiency 58.6%)
- **Note**: This Shannon coding is generally suboptimal compared to Huffman coding; we'll see better Huffman codes next.
- As $n \to \infty$: $L_n/n \to 0.469$ bits/symbol (100% efficiency)

2.11 Code Efficiency and Redundancy Analysis

2.11.1 Performance Metrics

Definition

For a code C with expected length L coding a source with entropy H:

$$\text{Efficiency: } \eta = \frac{H}{L} \times 100\% \quad \text{Redundancy: } \rho = L - H$$

2.11.2 Theoretical Bounds

From Shannon's theorem:

$$\frac{H}{H+1} \leq \eta \leq 1 \quad \text{and} \quad 0 \leq \rho < 1$$

Example

Efficiency vs. Entropy:

H (bits/symbol)	Minimum η	Maximum ρ	Interpretation
0.1	9.1%	0.9 bits	Very compressible, but +1 term dominates
1.0	50%	1.0 bit	Worst-case Shannon bound for sources with $H = 1$
2.0	66.7%	1.0 bit	+1 becomes less significant
4.0	80%	1.0 bit	High entropy, good efficiency possible
7.0	87.5%	1.0 bit	+1 overhead relatively small

Table 3: Theoretical limits on code efficiency for different entropy values

2.12 Huffman Coding: Optimal Prefix Code Construction

2.12.1 The Huffman Algorithm: Step-by-Step

Theorem 4 (Huffman, 1952). *For a given source with symbol probabilities $p_1, p_2, \ldots, p_m$, the Huffman algorithm produces a prefix code that minimizes the expected code length*

$L = \sum p_i \ell_i$.

Algorithm 1 Huffman Code Construction (Complete Version)

Require: Symbols $x_1, \ldots, x_m$ with probabilities $p_1, p_2, \ldots, p_m$

Ensure: Optimal binary prefix code

1: Create a min-priority queue Q initialized with m nodes, each containing one symbol and its probability
2: **while** $|Q| > 1$ **do**
3: $a \leftarrow$ EXTRACT-MIN(Q) {Node with smallest probability}
4: $b \leftarrow$ EXTRACT-MIN(Q) {Node with next smallest probability}
5: Create new node z with $p_z = p_a + p_b$
6: Make a left child of z, b right child of z
7: INSERT(Q, z)
8: **end while**
9: The remaining node in Q is the root of the Huffman tree
10: Traverse tree from root to leaves, assigning 0 to left edges, 1 to right edges

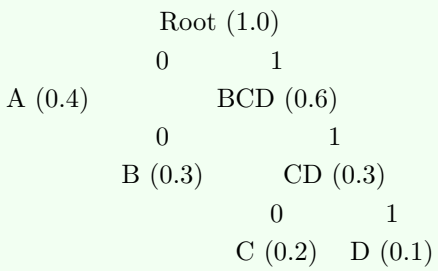
Example

Huffman Coding Example: Source with probabilities:

$$p(A) = 0.4, \quad p(B) = 0.3, \quad p(C) = 0.2, \quad p(D) = 0.1$$

1. **Step 1**: Combine C(0.2) and D(0.1) $\rightarrow$ CD(0.3)
2. **Step 2**: Combine B(0.3) and CD(0.3) $\rightarrow$ BCD(0.6)
3. **Step 3**: Combine A(0.4) and BCD(0.6) $\rightarrow$ Root(1.0)
4. **Codes**: A=0 (1 bit), B=10 (2 bits), C=110 (3 bits), D=111 (3 bits)

Huffman Tree Visualization:



Analysis:

- Expected length: $L = 0.4 \times 1 + 0.3 \times 2 + 0.2 \times 3 + 0.1 \times 3 = 1.9$ bits/symbol
- Entropy: $H = 1.846$ bits/symbol

- Efficiency: $\eta = 1.846/1.9 = 97.1\%$
- Shannon code (from earlier): $L = 2.4$ bits, $\eta = 76.9\%$
- **Huffman is significantly better than Shannon coding!**

2.12.2 Huffman vs. Shannon's Theorem

- **Shannon's Theorem**: Proves $H \leq L < H + 1$ is achievable
- **Shannon Coding**: Simple construction with $\ell_i = \lceil -\log_2 p_i \rceil$
- **Huffman Coding**: Optimal construction that minimizes L
- **Relationship**: $L_{\text{Huffman}} \leq L_{\text{Shannon}} \leq H + 1$
- **For dyadic sources**: Both achieve $L = H$ exactly

2.12.3 Huffman Code Properties

Theorem 5 (Huffman Code Length Bound)**.** *For a Huffman code with codeword lengths ℓ_i, each length satisfies:*

$$\ell_i \leq \lceil -\log_2 p_i \rceil$$

That is, no Huffman codeword is longer than the corresponding Shannon codeword.

- **Optimality**: Minimizes expected code length among all prefix codes
- **Uniqueness**: **Not unique in general**. If all probabilities are distinct and no ties occur during merging, the code lengths are unique up to relabeling; otherwise multiple optimal trees may exist.
- **Length bound**: $\ell_i \leq m - 1$ for m symbols
- **Two least probable**: Always have same length, differ only in last bit
- **Kraft inequality**: Huffman codes satisfy $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$, with equality if and only if the tree is complete

2.12.4 Optimal Code Structure Lemma

Lemma 1. *In an optimal prefix code:*

1. *If $p_j > p_k$, then $\ell_j \leq \ell_k$ (more probable = shorter code)*
2. *The two least probable symbols have the same length*
3. *The two least probable symbols differ only in the last bit*

2.12.5 Optimality Proof of Huffman Coding

Proof Sketch. By induction on number of symbols m:

Base case ($m = 2$): Trivial - need exactly 1 bit per symbol.

Inductive step: Assume Huffman optimal for $m - 1$ symbols.

Let x and y be two least probable symbols. Consider reduced alphabet where x and y are merged into z with $p_z = p_x + p_y$.

1. If C' is optimal for reduced alphabet, then creating C by splitting z into x and y (appending 0 and 1) gives:

$$L(C) = L(C') + p_x + p_y$$

2. Any optimal code for original alphabet must have x and y as siblings (same parent)
3. Huffman algorithm finds such sibling pairing
4. By induction hypothesis, C' is optimal for reduced alphabet
5. Therefore C is optimal for original alphabet

□

2.12.6 Redundancy Bound

Theorem 6 (Gallager's Redundancy Bound, 1978)**.** *For a Huffman code, the redundancy is bounded by:*

$$L - H < p_{min} + 0.086$$

where p_{min} is the smallest symbol probability. Moreover:

- *This bound is tight (achievable for some distributions)*
- *For dyadic sources ($p_i = 2^{-k_i}$): $L = H$ exactly*
- *Worst-case redundancy occurs when $p_{min} \approx 0$*

Example

Redundancy Analysis:

- **Best-case**: For dyadic source with $p_i = 2^{-k_i}$, $L = H$ exactly
- **Worst-case**: When smallest probability is very small
- **Example**: For source with probabilities $\{0.999, 0.001\}$:
  - Huffman codes: 0→0, 1→1 (1 bit each)

- $L = 1$ bit/symbol, $H \approx 0.011$ bits/symbol
- Redundancy: $\rho \approx 0.989$ bits/symbol
- Efficiency: $\eta \approx 1.1\%$

2.13 Extended Huffman Coding: Approaching the Entropy Limit

2.13.1 The Problem with Basic Huffman

Even optimal Huffman coding suffers from the "+1" term in Shannon's theorem:

$$L < H(X) + 1$$

For low-entropy sources, this overhead is significant:

Example

Binary source with $p(0) = 0.9$, $p(1) = 0.1$:

- Entropy: $H(0.9) = 0.469$ bits/symbol
- Basic Huffman: Symbols {0,1}, codes {0,1}, $L = 1$ bit/symbol
- Efficiency: $\eta = 46.9\%$ (poor!)
- Problem: Overhead 0.531 bits/symbol ¿ entropy itself!

2.13.2 Block Huffman Coding Solution

Code n symbols together as "super-symbols":

1. Consider nth extension of source: $X^n = (X_1, X_2, \ldots, X_n)$
2. Alphabet size grows to m^n sequences
3. Apply Huffman coding to these blocks

Theorem 7 (Extended Huffman Performance)**.** *For the nth extension coded with Huffman, the average length per symbol satisfies:*

$$\lim_{n\to\infty} \frac{L_n}{n} = H(X)$$

Note: While Huffman coding typically performs at least as well as Shannon coding, it lacks a universal closed-form finite-n bound like $L_n/n < H + 1/n$.

Example

Extended Huffman for Binary Source $p(0) = 0.9$**,** $p(1) = 0.1$:
$n = 2$ **blocks (4 sequences)**:

- Probabilities: $P(00) = 0.81$, $P(01) = 0.09$, $P(10) = 0.09$, $P(11) = 0.01$
- Huffman codes: 00→0, 01→10, 10→110, 11→111
- Expected length: $L_2 = 0.81 \times 1 + 0.09 \times 2 + 0.09 \times 3 + 0.01 \times 3 = 1.29$ bits/block
- Per symbol: $L_2/2 = 0.645$ bits/symbol
- Efficiency: $\eta = 0.469/0.645 = 72.7\%$ (vs 46.9% for $n = 1$)

$n = 3$ **blocks (8 sequences)**:

- Probabilities:

$$\begin{aligned} &000 : 0.729, \quad 001 : 0.081, \quad 010 : 0.081, \quad 011 : 0.009 \\ &100 : 0.081, \quad 101 : 0.009, \quad 110 : 0.009, \quad 111 : 0.001 \end{aligned}$$

- **Note**: Showing optimality for large n is complex. The following is illustrative, not proven optimal.
- One possible Huffman assignment:

$$\begin{aligned} &000 \to 0, \quad 001 \to 100, \quad 010 \to 101, \quad 011 \to 11000 \\ &100 \to 1101, \quad 101 \to 11001, \quad 110 \to 1110, \quad 111 \to 1111 \end{aligned}$$

- Expected length: $L_3 \approx 1.6$ bits/block
- Per symbol: $L_3/3 \approx 0.533$ bits/symbol
- Efficiency: $\eta = 0.469/0.533 \approx 88.0\%$

Trend: As n increases, $L_n/n \to H(X)$

2.13.3 Practical Issues with Block Huffman

- **Exponential complexity**: Alphabet size grows as m^n
- **Memory requirements**: Need to store $2m^n - 1$ nodes in Huffman tree
- **Delay**: Must wait for n symbols before encoding/decoding

- **Adaptation**: Probabilities may change over time

Important

Warning: Theoretical vs. Practical Use of Block Huffman Block Huffman coding demonstrates that we can approach the entropy limit arbitrarily closely, but it is **of theoretical interest only for large n**. The exponential growth in complexity makes it impractical for real applications with large alphabets or large block sizes.

Example

Complexity Growth for English Text ($m = 256$):

n	**Block Size**	**# Sequences**	**Tree Nodes**
1	1 byte	256	511
2	2 bytes	65,536	131,071
3	3 bytes	16.7 million	33.5 million
4	4 bytes	4.3 billion	8.6 billion
5	5 bytes	1.1 trillion	2.2 trillion

Table 4: Exponential growth makes large n impractical

2.14 Adaptive Huffman Coding: Real-time Solution

2.14.1 The FGK Algorithm (Faller-Gallager-Knuth)

- **Dynamic**: Updates code as symbols arrive
- **No initial model**: Learns probabilities on-the-fly
- **Sibling property**: Maintains optimality after each update
- **Complexity**: $O(m)$ worst-case per update; can be reduced with careful data structures
- **Applications**: Used in early versions of UNIX compress, modem protocols

2.14.2 Comparison: Static vs Adaptive vs Block

Method	Efficiency	Complexity	Delay
Static Huffman	High if model good	$O(m \log m)$	None
Adaptive Huffman	Medium-High	$O(m)$ per symbol	None
Block Huffman (n)	Approaches optimal	$O(m^n \log m^n)$	n symbols

Table 5: Trade-offs in Huffman coding variants

Important

Why Arithmetic Coding Beats Huffman Huffman coding is limited by the **integer constraint** on codeword lengths, leading to the "+1" overhead in Shannon's theorem. Arithmetic coding, which we'll cover next, uses **fractional bits** and can achieve average lengths arbitrarily close to $H(X)$ without block coding, removing the "+1" penalty entirely.

2.15 Practical Implications and Limitations

2.15.1 Assumptions of Shannon's Theorem

- **Discrete Memoryless Source**: Symbols independent and identically distributed
- **Known Distribution**: Probabilities p_i are known in advance
- **Arbitrary Delay**: Block coding allows infinite delay for encoding/decoding
- **No Complexity Constraints**: No limits on computational resources

2.15.2 Violations in Practice

Example

Real-world Violations:

- **Dependencies**: English text has strong correlations between letters
- **Unknown Distribution**: Must estimate probabilities from data
- **Delay Constraints**: Real-time applications limit block size
- **Complexity**: Exponential growth with block size (m^n sequences)
- **Solution Approaches**: Adaptive coding, dictionary methods (LZ), arithmetic coding

2.16 Extensions and Generalizations

2.16.1 Markov Sources

For a kth order Markov source with conditional entropy $H(X|X^k)$, the theorem extends to:

$$H(X|X^k) \leq L < H(X|X^k) + 1$$

2.16.2 Universal Coding

When the source distribution is unknown, universal codes achieve:

$$\frac{1}{n} L_n \to H(X) \quad \text{almost surely}$$

Examples: Lempel-Ziv codes, arithmetic coding with adaptive models.

2.16.3 Rate-Distortion Theory

For lossy compression with distortion D, the rate-distortion function $R(D)$ gives the minimum achievable rate:

$$R(D) = \min_{p(\hat{x}|x):\mathbb{E}[d(X,\hat{X})] \leq D} I(X;\hat{X})$$

2.17 Advanced Examples and Applications

Example

DNA Sequence Compression: Alphabet $\{A, C, G, T\}$ with typical probabilities $\{0.3, 0.2, 0.2, 0.3\}$

- Entropy: $H = 1.97$ bits/base
- Simple code: 2 bits/base (efficiency 98.5%)
- Exploiting dependencies: Adjacent bases are correlated in genomes
- Conditional entropy: $H(X_n|X_{n-1}) \approx 1.5$ bits/base
- Practical compressors achieve 1.6 bits/base

Example

Image Compression Limit: Grayscale image with 256 levels

- Naive: 8 bits/pixel
- Actual entropy from pixel correlations: Typically 1-4 bits/pixel

- PNG (lossless): 2-6 bits/pixel (uses filtering + DEFLATE = LZ77 + Huffman)
- JPEG (lossy): 0.5-2 bits/pixel with visual quality
- Theoretical limit from image statistics

Historical Notes

- **1948**: Claude Shannon publishes "A Mathematical Theory of Communication"
- **1949**: Leon Kraft proves inequality for prefix codes
- **1952**: David Huffman, as a graduate student at MIT, invents optimal prefix coding algorithm
- **1956**: Brockway McMillan extends Kraft inequality to all uniquely decodable codes
- **1978**: Robert Gallager proves tight redundancy bound for Huffman codes

Homework Assignment 2: Source Coding Theory and Huffman Coding

Note: Questions 2 and 3 are comprehensive problems. Question 3 is a challenge problem that explores block coding in depth.

1. **Kraft-McMillan Applications (20 points)**
   (a) Prove that for any uniquely decodable code with codeword lengths $\ell_1, \ell_2, \ldots, \ell_m$, we have $\sum_{i=1}^{m} 2^{-\ell_i} \leq 1$
   (b) Given lengths $\{2, 3, 3, 3, 4, 4\}$, verify if a binary prefix code exists
   (c) If a code exists, construct it using the canonical method
2. **Huffman Code Construction (30 points)**
   (a) For source with probabilities $\{0.35, 0.2, 0.15, 0.1, 0.1, 0.05, 0.05\}$:
      - Construct Huffman tree step by step (show all intermediate steps)
      - Assign codewords and calculate expected length
      - Compute efficiency and redundancy
   (b) Using the lemma from class, prove that in your Huffman code, the two least probable symbols have codewords of equal length

(c) Verify that your code satisfies the Kraft inequality (note: it may not satisfy it with equality if the tree is not complete)

3. **Extended Huffman Coding: Challenge Problem (25 points)**

   (a) Consider a binary source with $p(0) = 0.8$, $p(1) = 0.2$

   (b) Design Huffman codes for $n = 1, 2, 3$ (code blocks of symbols together)

   (c) For each n, calculate:

   - Expected length per block L_n
   - Expected length per symbol L_n/n
   - Efficiency η_n

   (d) Plot L_n/n vs n (for $n = 1, 2, 3$) and compare to the entropy limit

   (e) Explain mathematically why efficiency improves with n

4. **Optimality Analysis (15 points)**

   (a) For the source in question 2, design a Shannon code ($\ell_i = \lceil -\log_2 p_i \rceil$)

   (b) Compare the Huffman code with the Shannon code:

   - Expected lengths
   - Efficiencies
   - Redundancies

   (c) Under what conditions (what type of probability distributions) does Huffman coding achieve exactly $L = H$?

5. **Practical Considerations (10 points)**

   (a) Explain why block Huffman coding with large n is impractical for real applications

   (b) What alternative approaches exist for approaching the entropy limit without exponential complexity?

   (c) How does arithmetic coding (to be covered in the next lecture) solve the integer constraint problem?

Reading Assignment and References

- **Required Reading**:
  - Cover & Thomas, *Elements of Information Theory*, Chapter 5: Data Compression
  - Shannon, C. E. (1948). "A Mathematical Theory of Communication"

- **Advanced References**:
  - Huffman, D. A. (1952). "A Method for the Construction of Minimum-Redundancy Codes"
  - Gallager, R. G. (1978). "Variations on a Theme by Huffman"
- **Online Resources**:
  - Visual Huffman tree generator: `https://www.csfieldguide.org.nz/en/interactives/huffman-tree/`
  - Kraft inequality calculator: `https://planetcalc.com/8157/`